Bolt Beranek and Newman Inc.

LEVEL III

(12)

Report No. 4666

A099873

AD A103217

Development of a Voice Funnel System

Quarterly Technical Report No. 9
1 August 1980 to 31 October 1980

August 1981

Prepared for:
Defense Advanced Research Projects Agency

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 4666 — 2. GOVT ACCESSION NO. AD-A103217	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) Development of a Voice Funnel System, Quarterly Technical Report No. 9, (9) 1 Aug — 31 Oct 80 (14)	5. TYPE OF REPORT & PERIOD COVERED Quarterly Technical
	6. PERFORMING ORG. REPORT NUMBER BBN-4666
7. AUTHOR(s) (10) R. D. Rettberg	8. CONTRACT OR GRANT NUMBER(s) (15) MDA903-78-C-0356 ARPA Order-3653
9. PERFORMING ORGANIZATION NAME AND ADDRESS Bolt Beranek and Newman Inc. 50 Moulton Street, Cambridge, MA 02238	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS ARPA Order No. 3653
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Blvd., Arlington, VA 22209	12. REPORT DATE (11) Aug 1981
	13. NUMBER OF PAGES 28
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) (12) 33	15. SECURITY CLASS. (of this report) Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Voice Funnel, Digitized Speech, Packet Switching, Butterfly Switch, Multiprocessor

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This Quarterly Technical Report covers work performed during the period noted on the development of a high-speed interface, called a Voice Funnel, between digitized speech streams and a packet-switching communications network.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE UNCLASSIFIED 060100

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Report No. 4666 Bolt Beranek and Newman Inc.

DEVELOPMENT OF A VOICE FUNNEL SYSTEM

QUARTERLY TECHNICAL REPORT NO. 9
1 August 1980 to 31 October 1980

August 1981

This research was sponsored by the
Defense Advanced Research Projects
Agency under ARPA Order No.: 3653
Contract No.: MDA903-78-C-0356
Monitored by DARPA/IPTO
Effective date of contract: 1 September 1978
Contract expiration date: 30 November 1980
Principal investigator: R. D. Rettberg

Prepared for:

Dr. Robert E. Kahn, Director
Defense Advanced Research Projects Agency
Information Processing Techniques Office
1400 Wilson Boulevard
Arlington, VA 22209

Table of Contents

FIGURES

1. Introduction

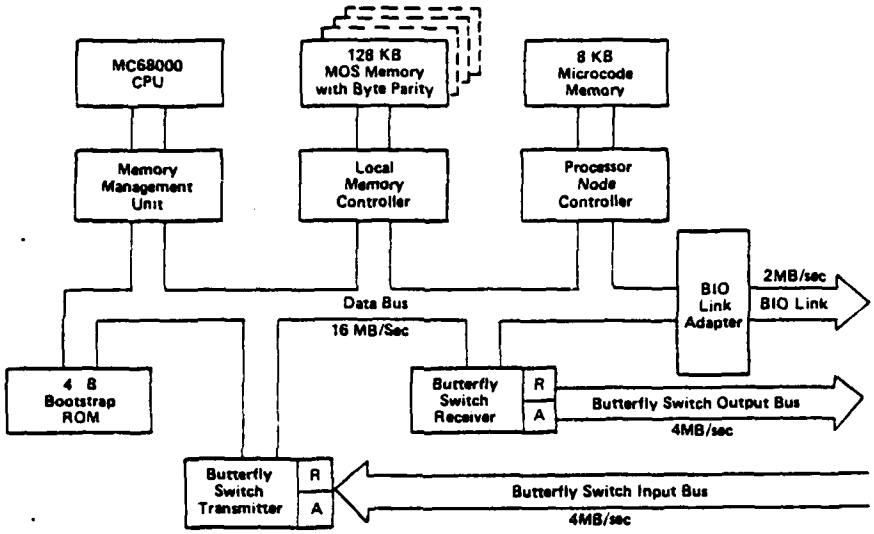
This Quarterly Technical Report, Number 9, describes aspects of our work performed under Contract No. MDA903-78-C-0356 during the period from 1 August 1980 to 31 October 1980. This is the ninth in a series of Quarterly Technical Reports on the design of a packet speech concentrator, the Voice Funnel.

This report describes the hardware design of the Butterfly Switch, which provides communication between Processor Nodes of the Butterfly Multiprocessor, the computer system upon which the Voice Funnel is constructed.

2. Butterfly Switch Message Processing

The Butterfly Multiprocessor consists of a number of Butterfly Processor Nodes connected together via a multi-level network of Butterfly Switch Nodes. The Processor Node hardware and the Switch Node hardware and topology have been discussed in the Design Report [Rettberg 79], and the Switch Node hardware was also described in Quarterly Technical Report Number 2 [Hoffman 79]. The reader is assumed to be familiar with Chapter 3 of the Design Report and with the general organization of the Processor Node hardware. In this report we discuss the actual message protocols which have been implemented and summarize the design changes made since the Design Report was written. Most of the changes are minor and easily explained, but the area of deadlocks and flow control has changed enough so that we have included a new section dealing with these issues.

The Butterfly Switch is critical to the operation of the Butterfly Multiprocessor, since all communications between Processor Nodes, including many processor-memory references, must be performed by the Butterfly Switch. The current design of the Processor Node's interface to the Butterfly Switch was described in general terms in a previous report [Rettberg 80]. The present report provides a far more detailed description of the Switch and its transactions.



Processor Node Components
Figure 1

The processor nodes are organized as shown in Figure 1. All switch transactions are initiated by the Motorola MC68000 CPU; an MC68000 service request causes micro-interrupt code in the Processor Node Control (PNC) to transmit an appropriate message using the Butterfly Switch Transmitter; this message is routed through a number of Switch Nodes, and finally appears at the Butterfly Switch Receiver in the destination processor. There the receiver causes microinterrupt code in the PNC to take the action specified in the message, which may include sending one or more additional messages to various other processors. In some cases the MC68000 that made the request waits for one of these responses to complete the transaction, while in other cases it simply initiates the transaction and proceeds immediately. Timers are used to recover from certain error conditions.

A variety of message transactions are provided. Each message includes at least the address of its destination processor, its message type, some data bytes, and a checksum. There are several different classes of messages: fixed vs. variable length messages, messages initiated by the MC68000 vs. those initiated by other messages, and messages which may initiate other messages vs. those which will not. These distinctions are important for an understanding of the way potential deadlocks have been avoided and of how the hardware is used.

The Processor Node implements these types of switch transactions: single word/byte reads or writes, block transfers, interrupt requests, Processor Node resets, and a class of special transactions which includes event synchronization, queueing and dequeueing, etc.

The single word transactions are initiated by the memory mapping hardware; that is, the memory mapping hardware maps ordinary MC68000 memory accesses into physical addresses which refer to a specific location in a specific processor via the switch. Since all memory accesses are handled by the PNC, these remote accesses appear no different from ordinary local accesses except for their speed.

The MC68000 explicitly initiates the other transactions by storing the address of a parameter block in one of several special locations which the PNC recognizes. This causes the PNC microcode to check the parameters and send out the appropriate message(s). The receipt of certain messages from the switch can cause the PNC to store into main memory, read from main memory, update queues, mark processes runnable, etc.

At this time all but the special transactions have been specified, coded, and debugged. The nature of the special transactions has been specified, but some details remain to be worked out. The final specifications for the major special transactions must be done with the details of the operating

system in mind, as they must work hand-in-hand to provide an efficient real-time environment. As far as the switch is concerned, these special messages are similar to the messages which have already been implemented.

2.1 Design Changes

This section documents the design changes made since the Design Report and QTR No. 2 were written. In addition to the changes discussed in this section, significant changes have also occurred in the deadlock and flow control areas. They are described later.

2.1.1 Error Handling

Error detection and handling occurs in several ways:

- Each message includes a 4-bit checksum, which is generated and checked automatically.
- Variable length block transfer data messages include an additional checksum early in the message, just after the address and length information.
- If alternate paths are available, rejected messages are automatically retried using the paths cyclically.
- Timers detect dead states for all messages and for the CPU when it is waiting for a reply.
- The application program or operating system may make additional checks as appropriate.

With these detection facilities, errors can be detected in the receiver, the transmitter, the PNC, and the CPU.

A four bit checksum in each message will detect most errors, but a given error will not be detected with a probability of 6%. This means that if errors are frequent the hardware must be considered broken, taken out of service, and fixed. Although one could try to retransmit a message with a bad checksum, it is safer to declare the hardware broken and get it fixed before it introduces undetected errors into the system. With this philosophy, the checksum error handler aborts the transaction in progress and reports the error to the operating system at the destination (where the error is detected). The operating system will include monitoring code which will attempt to locate the failing hardware and to resume fault-free operation using a subset of processor and/or switch nodes. Summaries of these errors will help to diagnose the more complex types of failure.

In a switch with extra columns, alternate addressing paths are available. These paths can be enabled and disabled independently by the MC68000. If a path fails solidly, the switch will quickly and automatically retry using alternate paths as long as the message continues to be rejected. However, if the path makes data errors, the operating system will run a diagnostic to identify the failing path, log it, and disable the path until the hardware has been repaired.

2.1.2 Block Transfers

At the time of the Design Report, we planned to implement two types of block transfer. One would have transferred data from the originating node as a variable length write request; the other would have transferred data to the originating node via a variable length reply. These have been replaced by a single, more general transaction in which a node (the originating node) may request the transmission of a block from any node to any node. There are now no restrictions on the locations of the source, destination, or originating node.

Because system latency requirements are proving easy to achieve, we are able to increase the maximum block size from 16 words to (approximately) 500 words. In the Voice Funnel this will be big enough to handle all normal data block transfers as a single operation. If longer transfers are required, the MC68000 software must break up the blocks into smaller transfers.

As a result of the increased maximum block size, we were able to simplify the block transfer by removing from the micro-machine the capability of breaking up block transfers which are too large into a series of smaller transfers. Since the maximum block transfer size has been enlarged and is, in fact, application dependent, it is more natural to perform this function at the application code level in the central processor.

Long messages have the advantage of reduced set-up time and contention. There is no real advantage to using shorter messages except for reduced latency. Even with 16 word messages it would not have been feasible to fit the entire message in a hardware buffer. The originating processor node becomes free as soon as the block transfer request has been accepted by the source processor node; however, accesses to either the source or destination processor nodes during the transfer are likely to be rejected. Error performance is not affected, since all errors indicate broken hardware. I/O DMA transactions are also not affected, since they have priority over block transfers, but MC68000 performance is reduced in the source and destination processors. In the absence of other activity the block transfer will use 75% of the total memory bandwidth, leaving 25% for the central processor.

2.1.3 Switch Performance and Flow Control

Instead of running the switch at 12 MHz and the CPU at 8 MHZ, we have chosen to run both at 8 MHz (a 125 nanosecond clock interval). There are at least two reasons for this change. One is the engineering difficulties involved in using two different clock speeds and designing the transmitter and receiver micro-machines to cope with this mismatch. Another is the amount of available memory bandwidth. The current switch design provides a maximum bandwidth of 32 Mbps point-to-point, while the current

memory system provides a maximum bandwidth of about 40 Mbps. This seems to be a good match and leaves a small amount of memory bandwidth available for other uses. The switch hardware itself could be changed to run at 64 Mbps, but this would not improve overall performance, due to the design of the rest of the system.

Although the Design Report states that we do not need flow control in the switch, we have since concluded otherwise. Two types of flow control are required, since the transmitting or receiving PNC may not be able to keep up with the switch at all times. This flow control operates at a low level and does not affect the CPU in any way. The details are discussed below.

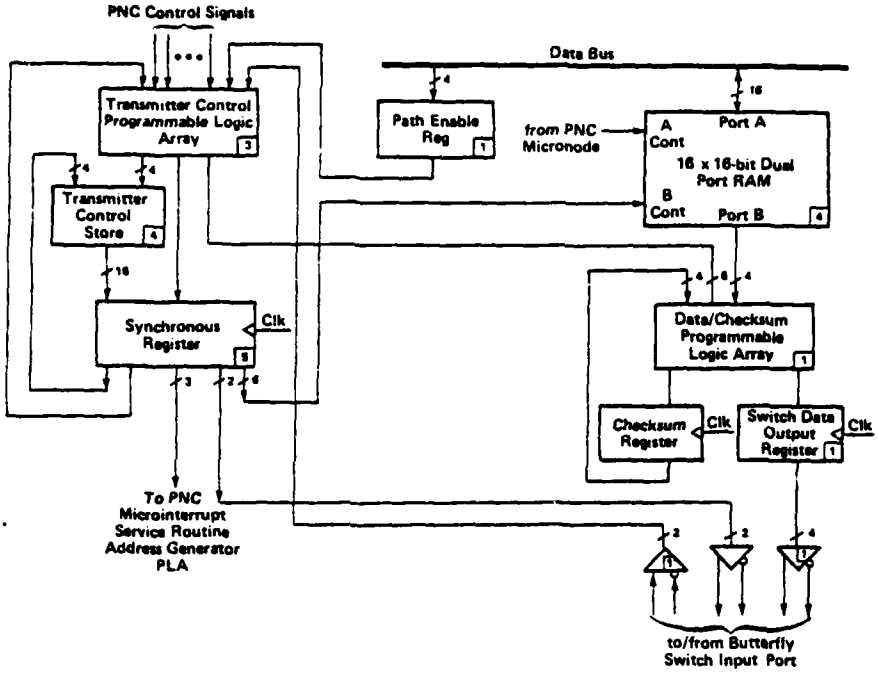
2.2 Message Processing

This section describes how messages are formatted, transmitted, and received. Such a description requires a fairly detailed explanation of how the hardware is organized. We therefore begin by describing the hardware elements and how they are used, and introduce the concepts and terminology required to discuss specific transactions. We then follow two example transactions through the system in order to illustrate the interactions involved. First we will consider a simple one-word read transaction, then a more complex example, the block transfer transaction.

2.2.1 Transmitter and Receiver Micro-machines

In order to transmit messages, the PNC makes use of an independent micro-machine, called the transmitter (see Figure 2). The transmitter communicates with the PNC via a 16-word dual ported memory called the TxRAM and via various control signals. The TxRAM is divided into two independent buffers. One of these, the request buffer, is used to initiate new transactions; the other, the acknowledgement buffer, is used to send secondary messages in response to messages coming in from the switch. Only one type of message may be going out at any one time, but a message of the other type may be prepared and stored in the TxRAM while the first is being transmitted.

All switch transactions originate in the MC68000, and are initiated by microcode in the PNC. No matter how complex the transaction, the first step is simple: a message is transmitted which requests the receiving PNC to take some action. Before this can happen the previous message of this type must have completed; in other words, the request buffer must be empty. The PNC will wait for the request buffer to become available. In the meantime it will continue to service non-MC68000 micro-interrupt service requests, but the MC68000 will be idle. Once the request buffer comes free, the PNC starts to build a message in the TxRAM and signals the transmitter to start sending the message. Since the PNC is uninterruptable at this point, it can start transmission before the message is complete.



Transmitter Finite State Machine
Figure 2

From the transmitting Processor Node's point of view there are two kinds of request message: query messages which always wait for a reply message to be returned by the destination processor node, and control messages which do not. For query messages, the PNC sets a return address register in its micro-interrupt system and enables a micro-interrupt on end-of-transmission. For control messages, it releases the MC68000. In either case, it then sets a timer and returns to its normal idle loop.

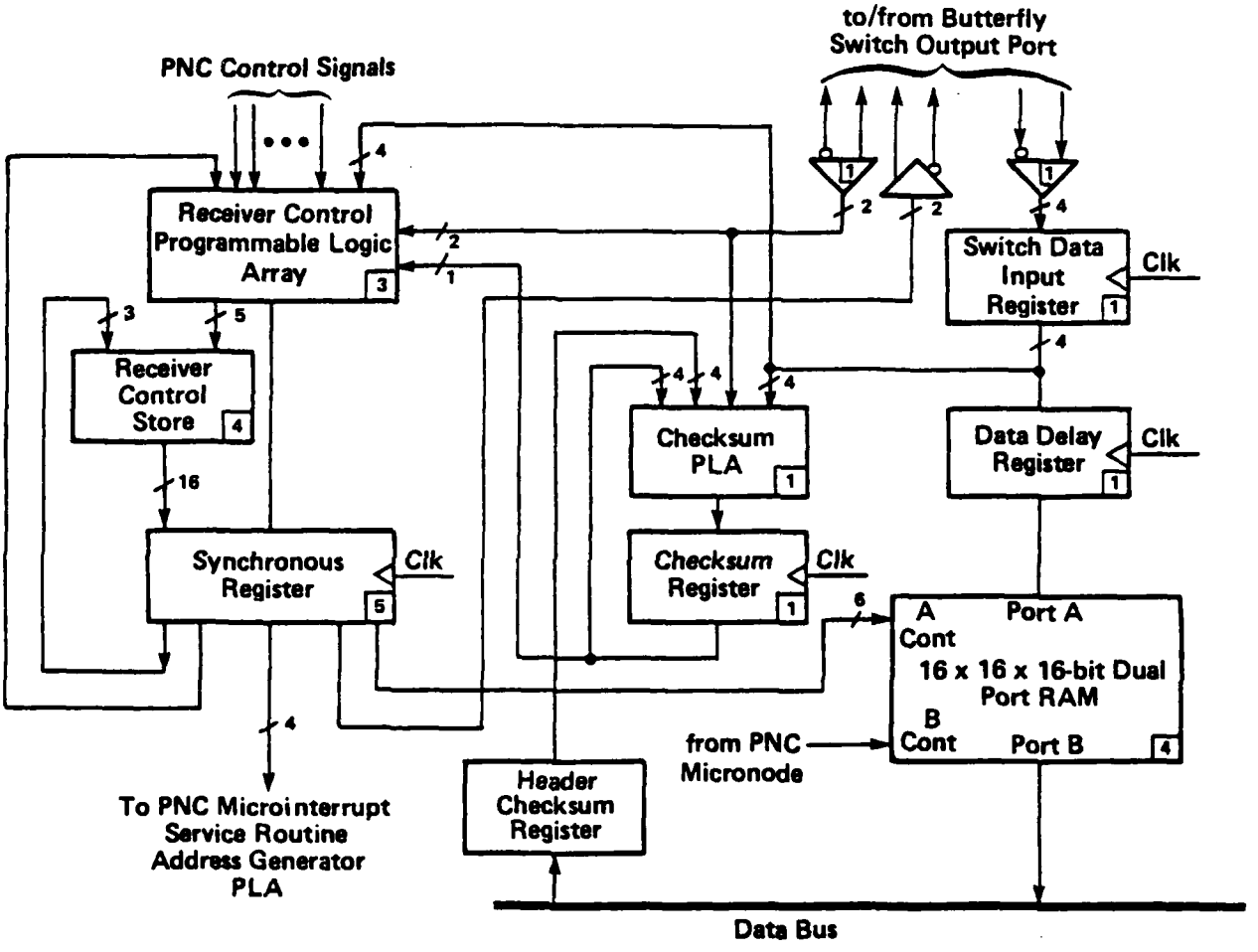
The transmitter microcode is now responsible for getting the message to the destination, if possible. Before it can start sending the message, it may have to finish sending an acknowledgement buffer message. When available, the transmitter then attempts to send this message out one of four alternate output paths. If the message is rejected, the transmitter automatically tries another path. Once the message header is accepted, the receiver may hold off the transmitter on a byte-by-byte basis. At the end of the message the transmitter sends a checksum.

When the message is completely transmitted, the transmitter terminates, requests a PNC micro-interrupt at the address specified in the return address register, and marks the request buffer available. The PNC resets the timer and micro-interrupt request, and, if a reply message is expected, sets the timer and return address register to wait for the reply message. If the

timer runs out, the PNC timer routine causes the transmitter to abort, notes the error, and simulates a transmitter completion.

To accept incoming messages, the PNC makes use of another independent micro-machine, called the receiver (see Figure 3). The receiver communicates with the PNC via a 16-word dual ported memory called the RxRAM and via various control signals. The RxRAM is divided into two independent buffers. One of these, the R-type buffer, is used to accept messages which may generate secondary messages; the other, the A-type buffer, is used to accept messages which can be processed entirely within the receiving Processor Node. Only a single message may be coming in at any one time, but a message of one type may be held in the RxRAM while a message of the other type is being received. The special message "reset" is accepted even if both receiver buffers are full, but all other messages are rejected if the appropriate buffer has not yet been emptied by the PNC.

R-type messages include all query messages and some control messages; A-type messages include other control messages plus the data messages discussed below. The reset control message needs no resources and is always processed immediately by the receiver itself, without intervention by the PNC and before the reset takes effect. R-type messages do not generate a PNC micro-interrupt and are therefore not processed by the PNC until the transmitter acknowledgement buffer is available. The R-type buffer becomes available as soon as the PNC has read the incoming



Receiver Finite State Machine
Figure 3

message, but the acknowledgement buffer is only released when the response to the previous R-type message is completely processed. The situation is simpler for A-type messages, which are processed immediately by the PNC. Both R-type and A-type messages are rejected by the receiver unless the appropriate buffer is empty.

During the processing of an R-type message the PNC may need to transmit one or more acknowledgement messages. Acknowledgement messages are broken into two types: reply messages which are sent directly in response to query messages, and data messages which are unsolicited. At the receiver both types go into the A-type buffer. The A-type buffer (and also the acknowledgement buffer) have two sections, the header section and the FIFO section. Messages which use these buffers may be short, using only the header area; they may be long, using both the header and the FIFO area; or they may be variable-length, using the FIFO dynamically. Variable length messages are used during the block transfer transaction, which is discussed below.

Each time the PNC assembles a message in the transmitter's request or acknowledgement output buffers, it initializes one of two timeout counters in order to detect deadlocks. Every 62.5 microseconds the memory refresh service routine increments both counters. If the request timeout counter reaches zero, the transmitter's request buffer is released, an error flag is set in the PNC status register, and processing proceeds as if the transmission had succeeded. Whenever a query-type request

message has been completely sent, a PNC micro-interrupt routine resets the request timeout counter to wait for the reply to be received. If there is no reply in the allotted interval, an error flag is set in the PNC status register, and a (query) type-dependent error routine is executed. If the acknowledgement timeout counter reaches zero, a flag is set in the PNC's micro-interrupt control register, and the transmitter's acknowledgement buffer is released with no further direct action. Thus if a transaction involves sending several acknowledgement messages, an attempt will be made to send all of them even if some time out; the PNC can test the bit to report whether previous acknowledgement messages have timed out.

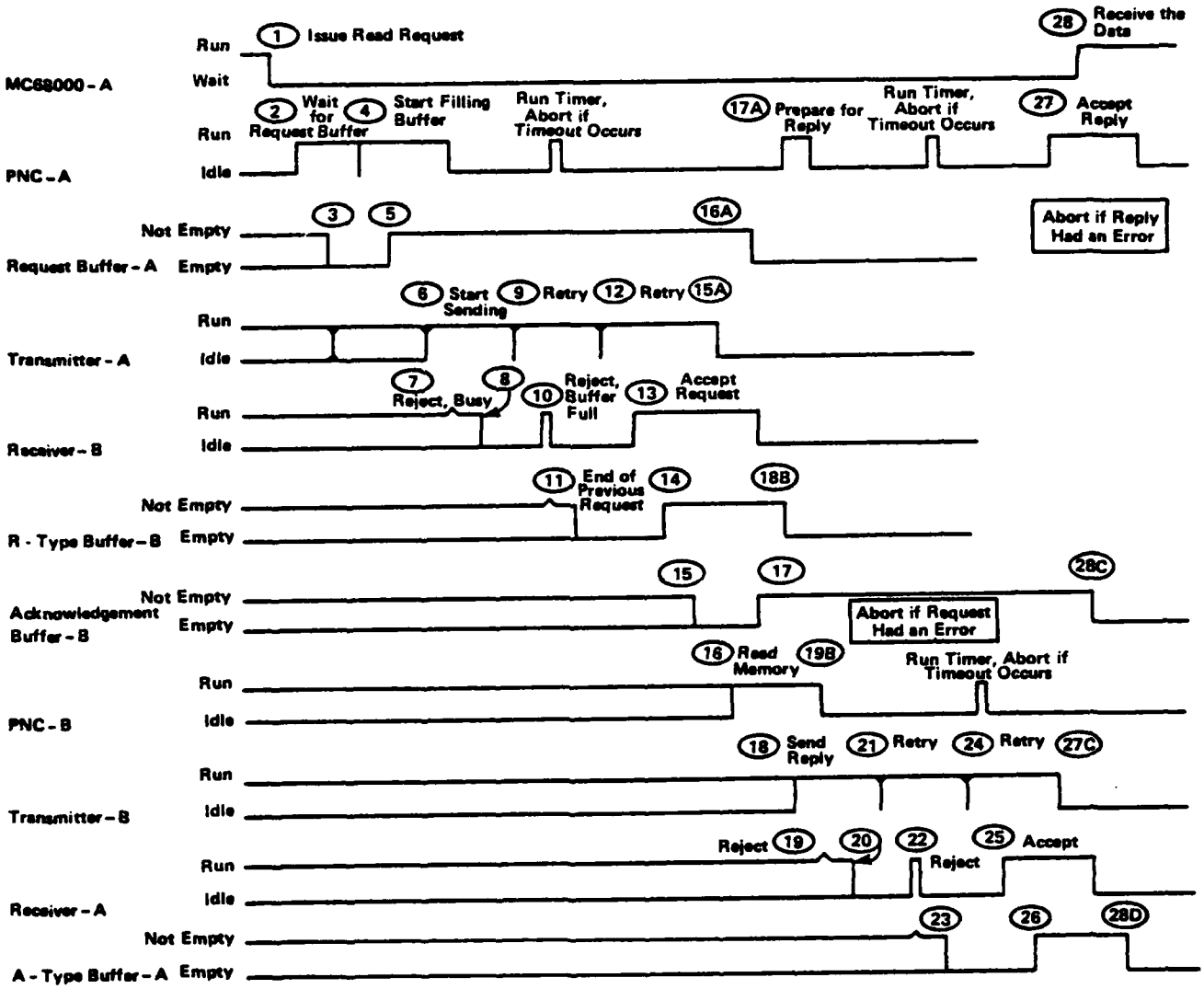
2.2.2 Example transactions

It is now possible to explain any of the fixed length transactions without introducing any significant new concepts. We will first examine a one word read transaction. The one word write is somewhat simpler, but it is almost identical to the first part of the block transfer transaction, which we will discuss below.

To provide an understanding of the one word read transaction we will examine its timing diagram, which shows the various elements, their actions at various times, and the events which trigger those actions. It is not possible, however, to

illustrate in one diagram every conceivable timing relationship, especially those involving errors, nor is it necessary to show the complete hardware state at every point. Figure 4 shows the details of a read transaction in which no errors occur. The description of the transmitter and receiver in the previous section should be detailed enough to allow the reader to understand the transaction, and also to understand how the scenario depicted in Figure 4 would be modified by the occurrence of errors.

The hardware elements involved are listed in the left-hand column; Processor Node A is shown performing a read from Processor Node B. The state of each processing element (run/wait, full/empty) is shown in the diagram. In cases where both states are indicated, it means the state is unknown; the actions shown are based on the worst case assumption that the unknown state is 'full', and the transaction must wait for the resource. We assume there is no contention in the switch itself. The numbering scheme associates state changes with the events which cause them. The first event is numbered one, 'Issue Read Request'. The other events are numbered to suggest the order in which they usually occur; where independent event sequences are triggered, the sequences are numbered in parallel with different letters appended to distinguish the sequences. For example, event 14 is followed by the independent event sequences 15, 16, ... and 15A, 16A, and 17A. Timer events occur every 62.5



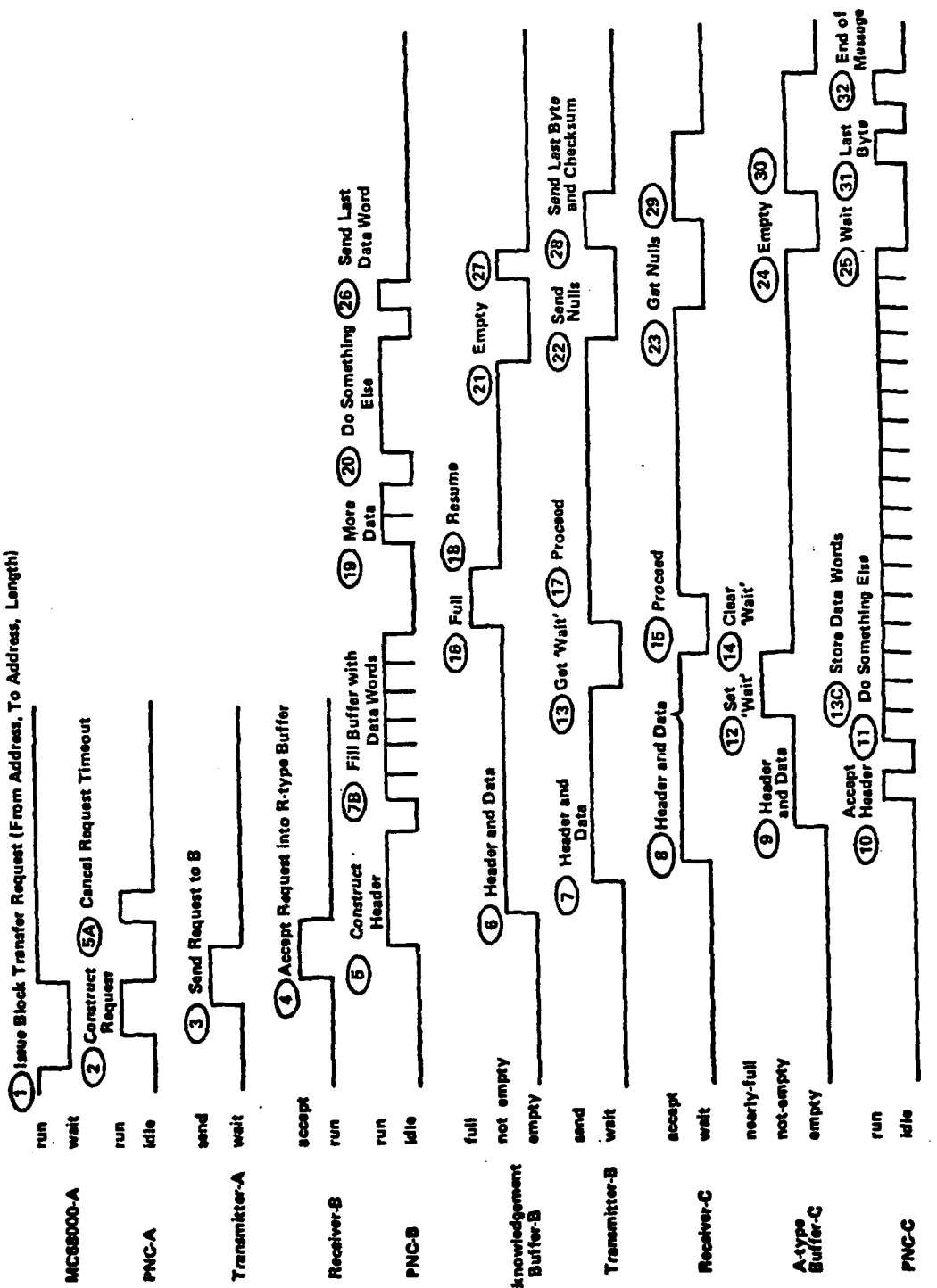
One-word Read Transaction -- Timing Diagram
Figure 4

microseconds in each PNC, asynchronously with switch transactions, and affect transaction processing only in case of errors.

This diagram does not try to give actual timing information, but only tries to show the order in which events occur and to suggest the pipelining involved. We show what happens when messages are rejected due to contention in the destination, but ignore the possibility of contention in the switch, which is handled in the same way. The likelihood of contention is load dependent, and there will frequently be no contention of any kind. In the absence of contention a one word read transaction takes about 3.1 microseconds longer than a normal local memory read.

Figure 5 illustrates a block transfer transaction, but in somewhat less detail. Processor Node A is shown requesting node B to send a block of B's memory to node C. We assume that all elements are empty or idle initially, and omit the state of A's request buffer and B's R-type buffer, which are similar to Figure 4.

The transmitter acknowledgement buffer has room for about 6 bytes of FIFO buffering; the receiver A-type buffer has room for about 14 bytes of FIFO buffering. If the receiver FIFO gets too full, the receiver requests the transmitter to wait (send nulls); if the transmitter FIFO empties, the transmitter sends nulls



Block Transfer Transaction
Figure 5

unilaterally. Each word of data causes a PNC micro-interrupt in both nodes A and B. These micro-interrupts occur only if there is room for more data in the transmitter buffer, or data ready to be removed from the receiver buffer. A separate receiver micro-interrupt initiates checks for various errors after all the data bytes have been stored. There is an optional micro-interrupt after a variable length message has been transferred, which is not currently used for block transfers, but which would allow multiple data messages or replies to be sent out.

It takes the PNCs 3 microcycles to process a word, while the transmitter and receiver take 4 microcycles each. That means that 25% of the PNC bandwidth is available for other tasks, such as performing memory accesses for the MC68000. Since I/O DMA and switch transfers have priority over MC68000 requests, it is possible that the MC68000s may get no service at all, even though the I/O system limits itself to 50% of the total PNC bandwidth. Thus, too many long transfers can cause latency problems. Since the usage pattern for block transfers and the latency constraints for the MC68000 are application dependent, the application designer must consider these issues.

2.3 Deadlocks and Flow Control

In most communications systems flow control is required to prevent data from being lost, either initially while a

communications channel is being established, or later, if some processing element is unable to keep up with the system as a whole. Introducing flow control mechanisms typically generates two secondary problems: deadlocks and dead states. By dead states we mean system states which persist indefinitely. These can occur as a result of either hardware or software bugs, or as the result of design deficiencies. Dead states which are the result of system design are termed deadlocks. Our philosophy in the switch was to avoid deadlocks by careful design, and to time out all dead states caused by user error or hardware problems external to a working processor node. Most dead states caused by local hardware problems will also time out. The occurrence of these dead states is reported to the local operating system. All the software involved in processing switch transactions is implemented in microcode, and once debugged, is not subject to change.

In the Butterfly, dead states may arise for a number of reasons. Messages can be rejected due to switch contention (a type of flow control), and this rejection can lead to dead states in any of the following cases:

- The addressed hardware is missing, broken, or flooded by some external malfunction.
- The local receiver is broken or flooded.
- The local transmitter or part of the switch is broken.

Once a message has been accepted, an empty transmitter buffer or a full receiver buffer can cause nulls to be sent. Broken hardware can simulate these conditions, resulting in a dead state. Finally, a dead state can occur when a message which requires a reply is not processed due to a bad checksum or hardware error.

Three dead states are timed out using two independent timers. When a request is submitted to the transmitter, a timer ensures that it goes out within a reasonable period. If a reply is expected, the same timer is used to ensure that it arrives within a reasonable period. A second timer does the same for the transmission of acknowledgement messages. These timers are set long enough so that the probability that they might be exceeded by normal switch contention is negligible, but short enough so that the operating system can recover smoothly if they occur.

Deadlocks are possible whenever a resource is needed to complete a task, and the resource is already in use and cannot be freed until the task completes. We have avoided deadlocks in this design by providing two input buffers and two output buffers in each processor node, and by adopting rules for how these buffers are used. These rules are somewhat different from the ones described in the Design Report.

Messages received in the A-type input buffer can always be processed immediately by the PNC, independent of the state of the

other buffers. There are no switch related PNC wait states which cannot service the A-type buffer; thus there are no deadlocks involved in sending to the A-type buffer. These messages can be from either transmitter buffer. All MC68000 requests use the request buffer, and all responses to incoming messages use the acknowledgement buffer; as a result, these cannot conflict directly. Messages from the acknowledgement buffer may be sent only to the A-type buffer, so there are no deadlocks associated with sending them. Messages sent to the R-type buffer are not serviced until the acknowledgement buffer becomes free, which it must do. Messages sent from the request buffer go either to the A-type or to the R-type buffer, both of which have already been shown to be deadlock free. Thus all four types of message are deadlock free.

One of our early designs did have a deadlock problem, which was discovered during performance simulation studies. The dual buffer, fixed-purpose strategy outlined above solves this problem and provides for useful pipelining under heavy load. Chapter 3 of the Design Report discusses an interesting alternative approach, in which request messages hold open their path through the switch, and wait for a reply to be generated. We briefly considered this method late in the switch design phase, but rejected it for several reasons:

- It would have required significant hardware additions and changes.

- Its primary effect on performance (which would occur in the case of the one-word read transaction under heavy load) was hard to evaluate.

Although we have implemented a uni-directional switch, the bi-directional switch has its advantages, and should also be considered as a potential technique for use in other implementations. If this were done, it would be useful to compare the complexity and performance of the alternative schemes.

3. References

[Hoffman 79] Hoffman, M., "Development of a Voice Funnel System, Quarterly Technical Report No. 2", Bolt Beranek and Newman Inc., Report 4143, June 1979.

[Rettberg 79] Rettberg, R. et al., "Development of a Voice Funnel System: Design Report", Bolt Beranek and Newman Inc., Report 4098, August 1979.

[Rettberg 80] Rettberg, R., "Development of a Voice Funnel System, Quarterly Technical Report No. 6", Bolt Beranek and Newman Inc., Report 4563, November 1980.

DISTRIBUTION OF THIS REPORT

Defense Advanced Research Projects Agency
Dr. Robert E. Kahn (2)
Dr. Vinton Cerf (1)

Defense Supply Service -- Washington
Jane D. Hensley (1)

Defense Documentation Center (12)

USC/ISI
Dr. Danny Cohen (2)

MIT/Lincoln Labs
Dr. Clifford J. Weinstein (3)

SRI International
Earl Craighill (1)

Rome Air Development Center/RBES
Neil Marples (1)

Defense Communications Agency
Gino Coviello (1)

Bolt Beranek and Newman Inc.
Library
Library, Canoga Park Office
R. Bressler
R. Brooks
P. Carvey
P. Castleman
G. Falk
J. Goodhue
E. Harriman
F. Heart
M. Hoffman
M. Kraley
A. Lake
W. Mann
R. Rettberg
P. Santos
E. Starr
E. Wolf